1.Introduction

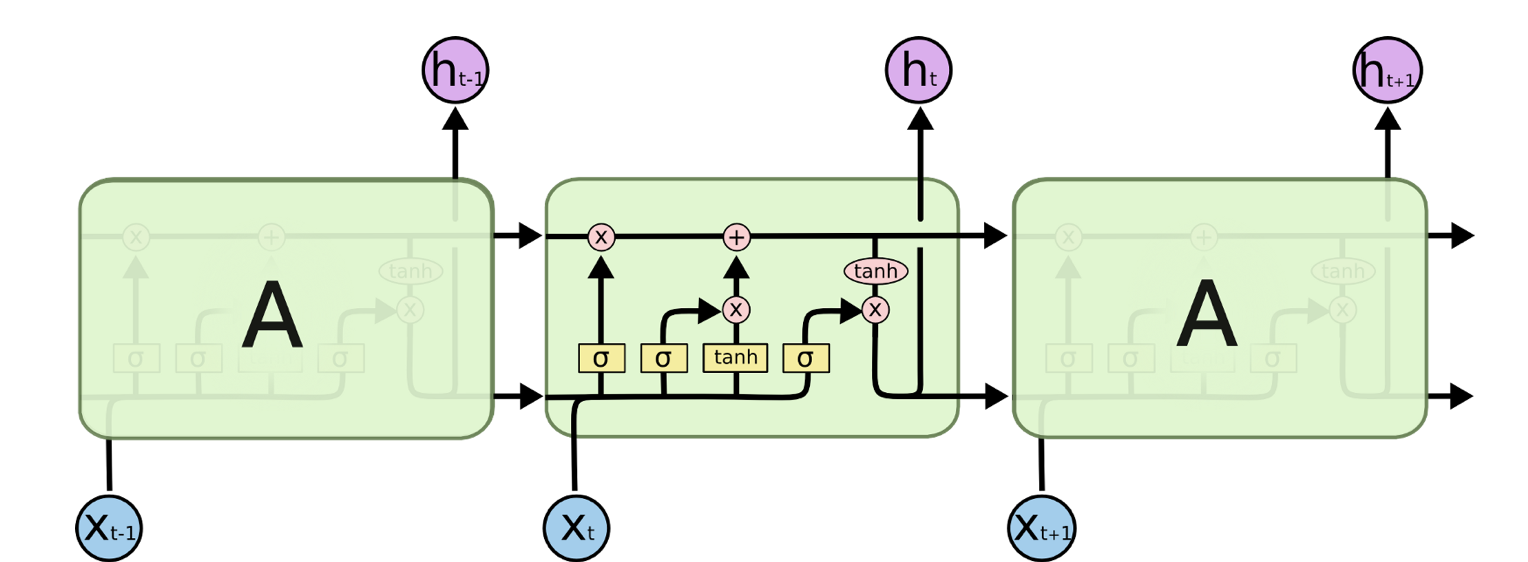
In this coursework, we want to do offensive sentences classification. Firstly, whether a sentence is an offensive one is distinguished by the first classifier. Then those offensive sentences is classified as targeted or untargeted. Finally, in the last stage, a 3-class classifier was used to judge whether the offensive sentence is targeted at an individual, a group or others.

The challenge is basically overcome in 2 steps. First of all, a predicational method of word embedding is used to convert the words into vectors, which takes also the context into account. The first stage can be realized by 2 ways. One is the skip-gram method, and the other is realized by the learnable layer of the nn.embedding(). Secondly, several classification strategies like DNN, CNN, LSTM, GRU and Transformer are tried to classify the sentences corresponding to Task A,B and C.

2.Methods

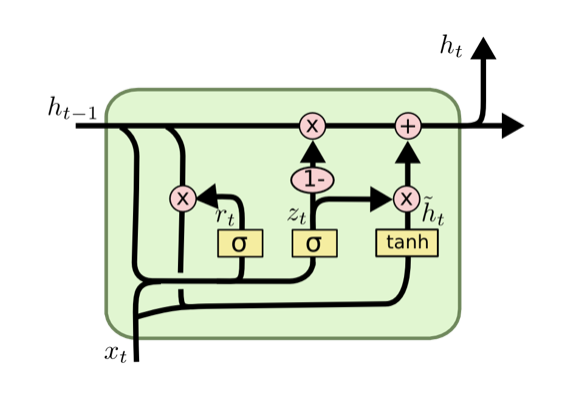
2.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory networks are a special kind of RNN, capable of learning long-term dependencies. They were introduced by in 1997[1]. The key idea of LSTM is to selectively remember the information got from former states by using gates of forget, input and output. In our case, each embedded word is an input xi.

 [2]

2.2 Gated Recurrent Unit (GRU)

A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit introduced in 2014[3]. It combines the forget and input gates together as a update gate, which can reduce the use of a parameter.

[2]

2.3 Bidirectional RNN

The idea of bidirectional RNN is proposed in 1997[4]. This idea can be used in both LSTM and GRU. The key idea is that we can both get information from a forward procedure of a sentence and also get some extra information when we train the network in a reverse direction. We will use the bidirectional LSTM(bLSTM) and bidirectional GRU(bGRU) in our excrement to get a higher classification accuracy.

3. Experiment design

3.1 Structure of LSTM

For task A, the pretrained weights of embedding is sent to the Embedding layer and keep it fixed. The embedding layer convert the size of words in a batch of 105-length sentences into 10(the embedding size of a word). Then the sentences are sent into the bLSTM layer. Then the first output h1(output size of 5) and the last output ht of the forward GRU and the first output h1\_bach and the last output ht\_back were sent into the next batch-normalization layer(4\*5=20). The batch normalization layer is used to reform the distribution of the output of bLSTM and hence increase the generalization ability and the trainability of the network. Finally, a dense layer of input size 20 and an output size 1 is used to get the classification result with a sigmoid computation after that.

LSTM(

(embedding): Embedding(15291, 10)

(lstm): LSTM(10, 5, bidirectional=True)

(bn2): BatchNorm1d(20, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(out): Linear(in\_features=20, out\_features=1, bias=True)

)

For Task B&C, the basic structure of the LSTM do not change expect the output size of the last layer. The parameter choice changes a to some degree for different tasks and the loss function of Task C changes from nn.BCELoss to nn.NLLLoss. Also, Task B&C face with an unbalance-dataset problem. For task B, the 0-1 ratio is about 1:5, so 2 experiments are done to explore the problem. Firstly, we remain the dataset unchanged. The classifier will be really biased towards the class 1 and get a low f1 score. The second approach we performed on task B is adding copies of class 0 data. This will lead to an increase of the f1 score. For task C, based on the result of task B, we directly perform weighted loss on the loss function (w = torch.Tensor([1.0,2.0,8.0]) loss\_fn = nn.NLLLoss(weight=w)), because the proportion of class 0,1,2 data is approximately 1:2:8.

3.2 Structure of GRU

The structure of bGRU is quite similar with above and similar tricks dealing with dataset are performed.

The pretrained weights of embedding is sent to the Embedding layer. The embedding layer convert the size of words in a batch of 105-length sentences into 10(the embedding size of a word). Then the sentences are sent into the bGRU layer. After that, the first output h1(output size of 8) and the last output ht of the forward GRU and the first output h1\_bach and the last output ht\_back were sent into the next batch-normalization layer(4\*8=32). The batch normalization layer is used as well. Finally, a dense layer of input size 32 and an output size 1. For Task B&C, the basic structure of the LSTM do not change expect the output size of the last layer.

GRU(

(embedding): Embedding(15291, 10)

(gru): GRU(10, 8, bidirectional=True)

(bn2): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(out): Linear(in\_features=32, out\_features=1, bias=True)

)

4. Results

3.3 Results of LSTM

Here we report the training accuracy, f1 score and confusion matrix on both training set and validation set. The training epoch step to stop and the parameters are manually tuned for each model to realize the best performance on validation dataset.

3.3.1 LSTM on Task A

Train accuracy: 80.15% | Valid acc: 76.96%

Train f1\_score: 0.74 | Valid f1\_score: 0.67

Valid confusion matrix:

[[868 34]

[271 151]]

3.3.2 LSTM on Task B

Before performing the 0-class data copy operation. The validation accuracy reached 88% but the classifier simply predicts every data as class 1. Copying the 0-class data sacrifices the validation accuracy to some degree but get a higher f1 score, which makes our classifier meaningful, since simply classifying everything to 1 make no sense in real life problem.

Train accuracy: 80.15% | Valid acc: 0.8540%

Train f1\_score: 0.57 | Valid f1\_score: 0.53

Valid confusion matrix:

[[ 12 94]

[ 52 842]]

3.3.3 LSTM on Task C

The f1score is calculated by ‘macro’ average of the 3 class

Train accuracy: Train accuracy: 65.33% | Valid acc: 66.24%

Train f1score: 0.48|Valid f1score: 0.53

Valid confusion matrix:

[[164 78 9]

[ 12 87 2]

[ 11 19 6]]

3.4 Results of GRU

GRU have less parameters which gives it 2 strength in comparison LSTM. Since our dataset is small, so less parameters make it easier to train the classifier. Also, it gives the model better generalization ability (We can observed overfitting after training several epochs of the classifier).

3.4.1 GRU on Task A

Train accuracy: 82.94% | Valid acc: 78.55%

Train f1\_score: 0.80 | Valid f1\_score: 0.74

Valid confusion matrix:

[[796 97]

[187 244]]

3.4.2 GRU on Task B

Train accuracy: Train accuracy: 81.79% | Valid acc: 84.49%

Train f1\_score: 0.54|Valid f1\_score:0.56

Valid confusion matrix:

[[ 20 86]

[ 74 820]]

3.4.3 GRU on Task C

Train accuracy: 66.54% | Valid acc: 70.10%

Train f1score: 0.50|Valid f1score: 0.54

Valid confusion matrix:

[[187 46 12]

[ 21 80 11]

[ 10 16 5]]

3.4.4 Comparing LSTM and GRU

As can be seen in the result of Task A&C GRU can perform slighted better than LSTM. This is mainly caused by the stronger generalization ability of the model. In task A, the main challenge is the generalization ability of our model since we can observe a 99% of training accuracy in our experiments but getting lower and lower accuracy of validation. That is the problem of overfitting. We basically tried 2 things to deal with the problem. One is organizing our dataset better, e.g. getting the stem of words, word embedding…The other is choosing a fewer-parameter model like GRU rather than LSTM like we covered in this section.

5. Conclusion

6. References

[1] Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.

[2] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[3] Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[J]. arXiv preprint arXiv:1406.1078, 2014.

[4] Schuster M, Paliwal K K. Bidirectional recurrent neural networks[J]. IEEE Transactions on Signal Processing, 1997, 45(11): 2673-2681.